Diseño y Análisis de Algoritmos
Trabajo Remedial

Jose Juan Arellano Juarez

Independent Set / Conjunto Independiente

Problema:

Tenemos un simple path que es una fila de nodos $v_1, v_2, ..., v_n$, y cada uno tiene un peso w_i. La regla del Conjunto Independiente dice que no puedes elegir dos nodos adyacentes. Si eliges el nodo 3, no puedes elegir ni el 2 ni el 4. Tu meta es sumar el mayor peso posible respetando esa regla.

(a): Proporcione la recurrencia que resuelve el problema de calcular el peso del conjunto independiente óptimo.

Para construir la recurrencia, usamos el razonamiento de ”in u out” / ”tomarlo o dejarlo”.
Definamos $OPT(i)$ como el peso máximo del conjunto independiente para el sub-camino que termina en el nodo i (es decir, considerando solo los nodos del 1 al i).
Para calcular $OPT(i)$, miremos el último nodo, v_i. Solo hay dos posibilidades de solución.

1. **Sí incluimos a** v_i: Si tomamos este nodo, ganamos su peso w_i, pero por la regla de independencia, no podemos tomar el nodo anterior (v_{i-1}). Por lo tanto, debemos sumar w_i al mejor resultado que obtuvimos hasta el nodo $i-2$.Valor: $w_i + OPT(i-2)$

- Valor: $w_i + OPT(i-2)$

2. **No incluimos a** v_i: Si no tomamos este nodo, entonces el peso máximo es simplemente el mejor resultado que podíamos obtener con los nodos anteriores (1 hasta $i-1$).

- Valor: $OPT(i-1)$

La ecuación formal se ve así:

$$OPT(i) = \max \begin{cases} OPT(i-1), & \text{(No incluimos } v_i\text{)} \\ w_i + OPT(i-2), & \text{(Síncluimos } v_i\text{)} \end{cases}$$

(b): Demuestre que su solución es óptima.

Casos Base:
Para que la recursión no sea infinita, necesitamos definir los primeros pasos:

- $OPT(0) = 0$ (Si no hay no dos, el peso es 0).
- $OPT(1) = w_1$ (Si solo hay un nodo, tomamos su peso).

Hipótesis de Inducción Fuerte:
Asumimos que nuestro algoritmo $OPT(i)$ calcula correctamente el peso máximo del conjunto independiente para cualquier camino de longitud i, donde $0 \leq i \leq k$.
Esto significa que damos por hecho que $OPT(k)$ es correcto Y que $OPT(k-1)$ también es correcto.

Paso Inductivo (Demostrar para $k+1$):
Objetivo: Demostrar que $OPT(k+1)$ calcula correctamente el peso óptimo para un camino con nodos $v_1, ..., v_{k+1}$.
Razonamiento: Sea S_{opt} una solución óptima real para el camino de tamaño $k+1$. Analicemos el último nodo, v_{k+1}. Solo hay dos casos posibles:

- **Caso A**: El nodo v_{k+1} SÍ está en S_{opt}. Entonces, el nodo anterior v_k no puede estar (por independencia). El resto de la solución S_{opt} - v_{k+1} debe ser la solución óptima para los nodos hasta $k-1$. Por nuestra Hipótesis, sabemos que nuestro algoritmo calcula esto correctamente como $OPT(k-1)$.

$$Peso(S_{opt}) = w_{k+1} + OPT(k-1)$$

- **Caso B**: El nodo v_{k+1} NO está en S_{opt}. Entonces, S_{opt} debe ser idéntico a la solución óptima para los primeros k nodos. Por nuestra Hipótesis, sabemos que nuestro algoritmo calcula esto correctamente como $OPT(k)$.

$$Peso(S_{opt}) = OPT(k)$$

Conclusión del paso: Como S_{opt} busca maximizar el peso, debe tomar el máximo de estos dos casos:

$$Peso(S_{opt}) = \max(OPT(k), \quad w_{k+1} + OPT(k-1))$$

Y esta es exactamente la fórmula que define nuestro algoritmo. Por lo tanto, el algoritmo es correcto para $k+1$

(c): A partir de la recurrencia del punto anterior, diseñe un algoritmo bottom-up que devuelva el peso del conjunto independiente óptimo.

Algorithm 1 Peso Máximo Independiente en un Camino

```
function PesoMaximoIndependiente(W[1..n])
    n ← longitud(W)                                ▷ Número de nodos del camino
    if n = 0 then
        return 0                                   ▷ Caso borde: grafo vacío
    end if
    Crear arreglo M[0..n]                          ▷ Tabla de memorización
    M[0] ← 0                                       ▷ Caso base: 0 nodos
    M[1] ← W[1]                                    ▷ Caso base: un solo nodo
    for i = 2 to n do
        M[i] ← max(W[i] + M[i − 2], M[i − 1])      ▷ Incluir o excluir el nodo i
    end for
    return M[n]                                    ▷ Resultado óptimo global
end function
```

(d): Utilice la información guardada en el arreglo de memorización para recuperar la identidad de los nodos que pertenecen a la solución óptima.

Algorithm 2 Reconstrucción Recursiva

```
function Sol(M[0..n], W[1..n], i)   ▷ Imprime/agrega los índices elegidos en la solución óptima hasta i
    if i = 0 then
        return                                     ▷ Caso base: no hay nodos que seleccionar
    end if
    if i = 1 then
        print 1                                    ▷ Caso base: se elige el único nodo
        return
    end if
    if W[i] + M[i − 2] > M[i − 1] then
        print i                                    ▷ Se incluye el nodo i
        Sol(M, W, i − 2)                           ▷ Se salta i − 1 por independencia
    else
        Sol(M, W, i − 1)                           ▷ No se incluye el nodo i
    end if
end function
```